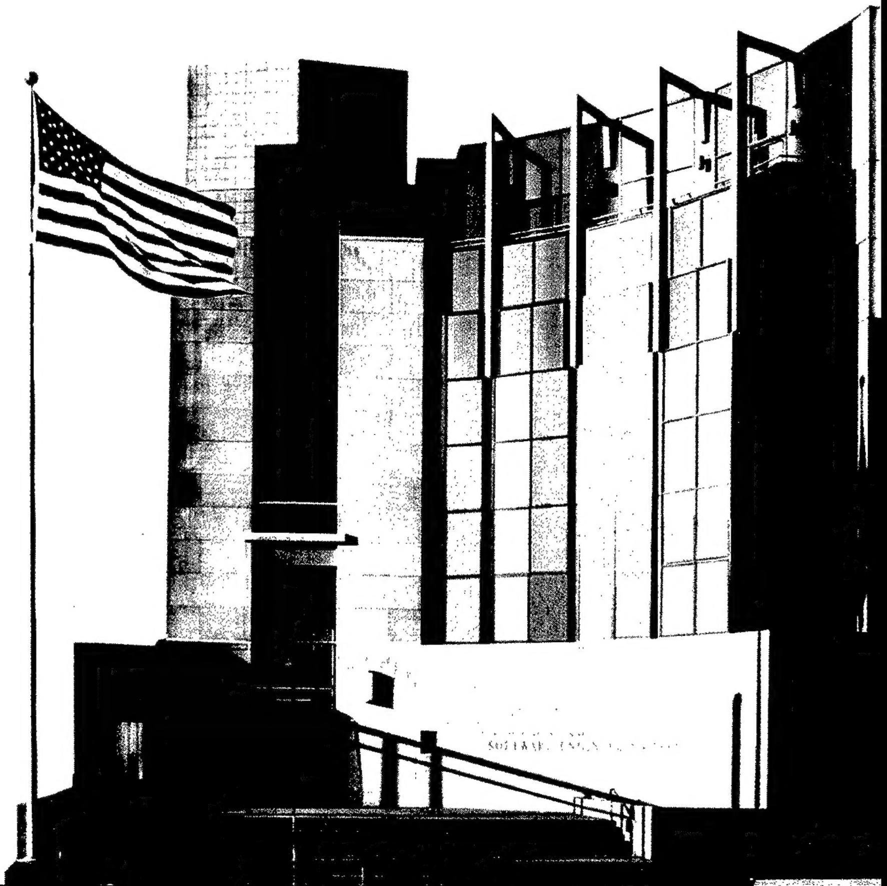
Carnegie Mellon
Software Engineering Institute

Salion, Inc.: A Software Product Line Case Study

Paul C. Clements
Linda M. Northrop

November 2002

TECHNICAL REPORT
CMU/SEI-2002-TR-038
ESC-TR-2002-038



Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Salion, Inc.: A Software Product Line Case Study

CMU/SEI-2002-TR-038
ESC-TR-2002-038

Paul C. Clements
Linda M. Northrop

November 2002

Product Line Practice Initiative

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

[illegible]

Christos Scondras
Chief of Programs, XPK

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Table of Contents

Acknowledgments v

Abstract vii

1 Introduction 1

2 Background 3
- 2.1 About the Company 3
- 2.2 Salion's Market 3
- 2.3 Salion's Product Line 5
- 2.4 Variabilities 6

3 How Salion Builds Its Software Product Line 9
- 3.1 Scoping 11
- 3.2 Understanding Relevant Domains 11
- 3.3 Process Definition 12
- 3.4 Architecture Definition 14
- 3.5 COTS Utilization 18
- 3.6 Tool Support 18
- 3.7 Marketing/Customer Interface Management 20
- 3.8 Operations 21
  - 3.8.1 Normal Operation 21
  - 3.8.2 Introducing New Variants 23
  - 3.8.3 Phasing Out Old Products 23
- 3.9 Data Collection, Metrics, and Tracking 23
- 3.10 Structuring the Organization 24

4 Payoffs and Benefits 27

5 Conclusions and Lessons Learned 29
- 5.1 Joyous Software Development Process, Part 1: Unified Vision 29
- 5.2 Joyous Software Development, Part 2: The Best of All Worlds 30
- 5.3 Customization Versus Configuration 31

5.4 Risk Management Culture 32

6 Conclusion **33**

7 For Further Reading **35**

References **37**

List of Figures

Figure 1: Salion, Its Customers, and Its Customers' Customers 4

Figure 2: Salion's Business Use Case Model 14

Figure 3: Salion's Product Line System Architecture 15

Acknowledgments

Case studies are always difficult to produce because they require inside information from people who are usually too busy, understandably, to spend time sharing with outsiders. Every successful case study is the result of extraordinary generosity on the part of the organization being covered. This one is no exception. This report would not have been possible without the time, energy, enthusiasm, and patience of the staff at Salion, Inc. In particular, we thank Ross Buhrdorf, Dale Churchett, Darwin Ling, Janis Cawthron, Bill Jones, and Susie Grandi. We also thank Charles Krueger, founder and chief executive officer (CEO) of BigLever, Inc., for his participation and insights and for the opportunity to see his company's Gears tool in action.

Abstract

This report is another in a series of Software Engineering Institute (SEI^{SM}) case studies of organizations that have adopted the software product line approach for developing systems. It details the story of Salion, Inc., an enterprise software company providing Revenue Acquisition Management solutions tailored to the unique needs of automotive suppliers. Salion's solutions enable suppliers to organize and manage their disparate customer-interfacing activities as one coordinated business process, resulting in higher revenues, profit margins, and customer satisfaction.

This case study is unique in that Salion did not have substantial experience in its application area, although its key designers and strategists were knowledgeable about related domains. Salion pursued a reactive approach to its product line that let it respond flexibly to spontaneous business opportunities and that significantly lowered the cost of adopting the product line paradigm to its software system development. This case study describes relevant dimensions of Salion's context, how it approached several product line practice areas that were key to its strategy, the benefits gained through its product line, lessons learned, and the major thematic aspects of the Salion story.

1 Introduction

This report is another in a series of Software Engineering Institute (SEI[SM]) case studies of organizations that have adopted the software product line approach for developing systems. A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [Clements & Northrop 02]. In contrast to one-at-a-time system development, software product lines epitomize strategic reuse. Using this approach, organizations are achieving order-of-magnitude improvements in time to market, cost, and product quality. In some cases, the flexibility and responsiveness brought about by the product line approach is allowing organizations to dominate their markets and achieve dominant entry positions in new markets.

This report details the story of Salion, Inc., an enterprise software company dedicated to helping suppliers optimize their revenue acquisition process.

This case study is unique in that Salion did not possess substantial experience in its application area, although its key designers and strategists were knowledgeable about related domains. This is in contrast to organizations like CelsiusTech [Brownsword & Clements 96], Market Maker [Clements & Northrop 02], Raytheon [Clements & Northrop 02], and Cummins, Inc. [Clements & Northrop 02] that had been building like systems for years before adopting the software product line approach. These and other organizations were able to parlay their in-depth knowledge and legacy artifacts into comprehensive up-front planning that let them define their product line scope and build their core asset base *a priori*. By contrast, Salion pursued a reactive approach [Krueger 01] to its product line that let it respond flexibly to spontaneous business opportunities and that significantly lowered the cost of adopting the product line paradigm to its software system development.

Section 2 begins by examining Salion's background, business area, and foray into software product lines. Section 3 examines how it approached several product line practice areas[1] that were key for it. Section 4 describes the payoffs as Salion gained more customers for its products. Section 5 concludes with lessons learned and a summary of the major thematic aspects of the Salion story.

SM SEI is a service mark of Carnegie Mellon University.

1 Twenty-nine product line practice areas categorized as software engineering, technical management, and organizational management are described by Clements and Northrop [Clements & Northrop 02].

2 Background

2.1 About the Company

Salion, Inc. is an enterprise software company dedicated to helping suppliers optimize their revenue acquisition process. Salion's initial focus is on suppliers who sell complex products via proposals. Its goals are to give those suppliers the power to target the best business, win new business, and deliver increased customer satisfaction.

Privately held, Salion was founded by McKinsey & Company and AV Labs (an early-stage venture fund) based on research conducted on how best to serve suppliers.

Salion entered its market knowing that the ability to deliver customized products quickly was going to be a prerequisite to its survival. In fact, its business model was based on the presumed ability to do this, even before the technical capability for achieving it was in place. Salion also knew that if it managed the products separately, the complexity would quickly overwhelm it and far outstrip the ability of its small staff to cope with the combinatorics of separately evolving systems. It realized that it needed to treat its products as separate versions of the same thing—in other words, as a software product line. Salion thus joined the small but growing body of companies that have adopted the software product line approach from the outset.

2.2 Salion's Market

Salion's customers are *suppliers*, manufacturers of fairly complex custom or semi-custom products. An example of a Salion customer is a Tier 1 automotive supplier that manufactures automotive interior assemblies or drive trains.[2] These products are designed and priced by the suppliers only after the product requirements are worked out with *their* customers (e.g., a major auto manufacturer). The suppliers do not typically sell off-the-shelf inventory built ahead of time.

Contracts between suppliers and their customers come about as a result of a negotiation protocol involving a request for a quote (RFQ) and a quote (or bid) in response. The bid is the

[2] In fact, Tier 1 automotive suppliers became Salion's first targeted market sector.

result of a collaboration within the supplier's organization that works out details of requirements, design, and pricing. The bid is the result of three factors:

1. the supplier's desire to go after the business represented in the RFQ
2. a design that satisfies the specifications
3. pricing that is high enough to allow the supplier to make money on the deal, should it receive the contract, and low enough to make its bid attractive

Figure 1 illustrates the two ways Salion delivers its software to its customers: hosted and installed. The hosted model removes the burden of maintenance from information technology (IT) departments while providing a secure and scalable environment. The installed model allows IT departments to incorporate Salion's software into their own IT infrastructure.

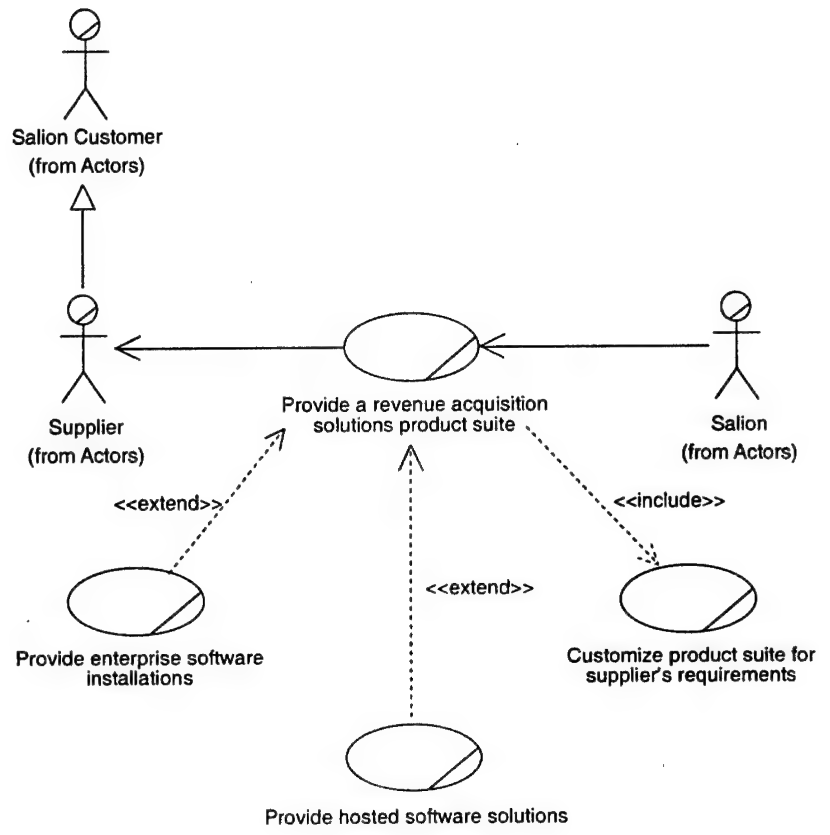

Salion Customer (from Actors)
Supplier (from Actors)
Provide a revenue acquisition solutions product suite
Salion (from Actors)
<<extend>>
<<include>>
<<extend>>
Provide enterprise software installations
Customize product suite for supplier's requirements
Provide hosted software solutions

Figure 1: Salion, Its Customers, and Its Customers' Customers

Salion markets a suite of software tools for Revenue Acquisition Management (RAM)—those activities associated with identifying valuable customer prospects, conducting sales efforts, and creating and delivering winning proposals. Suppliers succeed by winning proposals. However, the process of submitting proposals has traditionally been labor intensive and problematic. Producing a proposal takes time and considerable coordination. A late proposal will bar effective consideration, if not disqualify the company outright.

Typical problems in the traditional RAM processes include the inability to analyze bid requests to select the most promising opportunities, difficulty in keeping track of specification changes, no access to related organizational history, poor proposal tracking, inconsistent costing and pricing, and inefficiencies caused by manual document handling.

Salion's marketing literature includes some telling quotes that characterize the need and identify the resulting business opportunity:

> *"It should take us one day or less to turn a quote around. For some reason, it takes five weeks. This process is out of control."*—Director of Engineering, Tier 1 automotive supplier

> *"We recently rushed a late quote out the door that we thought we had priced with a 'nice margin'. In reality, the quote was for a part that we had been selling at twice the price we quoted. Luckily, our customer only asked for one year of retroactive rebates."*—Director of Sales, Tier 1 automotive supplier

> *"We just spent $100,000 on an opportunity that we had no chance of winning. We bid on the same business two years ago and our price was 50% too high. We have no way to capture or analyze our historical sales and bidding performance, so we make the same mistakes over and over."*—Tier 2 automotive supplier

> *"We spent $600,000 in overnight shipping costs last year."*—Tier 1 automotive supplier

2.3 Salion's Product Line

Salion's solution is a set of three software applications known collectively as the RAM Platform:

- Salion Revenue Process Manager™—This tool helps suppliers manage opportunities. It contains a workflow engine and Web-based communication tools to help a supplier organization manage the collaboration of design and pricing. It keeps track of a proposal's status and assists in the assembly of the final document.
- Salion Knowledge Manager™—This tool helps triage and analyze requests for proposals (RFPs), with decision support capabilities and analysis of bid performance, win/loss rates, and pricing. The idea here is to help a supplier analyze all available information to,

™ Salion Revenue Process Manager is a trademark of Salion, Inc.
™ Salion Knowledge Manager and Salion Business Link are trademarks of Salion, Inc.

among other things, choose the best candidates from among all the available opportunities. The Knowledge Manager uses historical information to prioritize opportunities and help the organization improve its response rates.

- Salion Business Link™—This tool extends collaboration between the supplier and the customer, and between the supplier and subsuppliers.

Each application is itself a suite of products. For example, Revenue Process Manager includes an opportunity manager, a proposal manager, and a change-order manager. Knowledge Manager includes a performance analysis manager, a triage analysis manager, and a resource analysis manager.

Salion is targeting large custom and semi-custom manufacturing organizations that rely on bids or proposals to win business that typically must deal with large numbers of engineering change orders in the course of doing business. The company's initial focus is on the automotive market, but other industries will also be targeted in the future. Estimates put the size of the automotive market at 19,000 suppliers representing an initial market opportunity of several billion dollars.

Because of the high stakes involved in proposal management, hosted and application service provider (ASP) customers demand the utmost confidentiality. As a result, security is one of the most important software quality attributes, in addition to performance and availability.

2.4 Variabilities

Salion products must work under a variety of contexts, compelling them to be variable with respect to the following:

- A single installation of the tool suite can have virtually any combination of the products described in Section 2.3.
- The software can run on the customer's hardware at the customer's site (i.e., installed), on Salion's hardware at Salion's site (i.e., hosted), or in combination with other customers' systems. This last model, called the ASP model, is discussed in Section 3.4.
- All customers will have a unique workflow, a unique set of input screens and other user-interface concerns, and a unique set of reports they want to generate.
- Each customer will have unique "bulk load" requirements, involving the transformation of existing data and databases into forms compatible with Salion's products.
- An automotive industry trade group (Open Applications Group [OAG]) has defined a business-to-business transaction framework encompassing some 120 standard objects to be used to transfer information from organization to organization. Because not every customer will need to use all 120 objects, this framework represents another up-front point of variation.

In addition to these predicted variations reflected in its architecture and business plan, Salion must also react to variations that it cannot foresee. For example, one customer may want the system to include a mobile phone interface.

3 How Salion Builds Its Software Product Line

A product line approach to software involves three essential activities: (1) development of core assets, (2) development of products, and (3) management of the product line operation at both technical and enterprise levels [Clements & Northrop 02]. The order in which core asset development and product development occurs is not fixed.[3]

Many organizations begin a software product line by developing the core assets. These organizations take a *proactive* approach [Krueger 01]. They define their product line scope to define the set (more often, a *space*) of systems that will constitute their product line. This scope definition provides a kind of mission statement for designing the product line architecture, components, and other assets with the right built-in variation points to cover the scope. Producing any product within that scope becomes a matter of exercising the variation points of the components and architecture—that is, configuring—and then assembling and testing the system. Other organizations begin with one or a few existing products and use them to generate the product line core assets and future products. Such organizations take a *reactive* approach.

The proactive approach has obvious advantages—products come to market extremely quickly with a minimum of code writing. But that approach also has disadvantages. It requires a significant up-front investment to produce the architecture and components that are generic (and reliable) across the entire product space. And it also requires copious up-front predictive knowledge—something that is not always available. In organizations that have long been developing products in a particular application domain, this is not a tremendous disadvantage. For a "green field" effort like Salion's, where there is neither experience nor existing products, this is an enormous risk.

The reactive approach has the advantage of a much lower cost of entry to software product lines, because the asset base is not built up front. However, the quality of the architecture and components of the product(s) used as the basis for the asset base must be robust, extensible, and appropriate to future product line needs. This is not a trivial prerequisite.

[3] Regardless of the order, there is perpetual iteration between these two activities throughout the life of the product line.

Krueger describes a form of the reactive model that works by building the first system (or a number of small systems) and, when a new product is defined for a new customer, "reacting" to it by refactoring its design to define what is now common across all members of the emerging product line and what varies [Krueger 01]. Variability among systems is handled, if practical, by making the system configurable with respect to that variability. Otherwise, the special need is treated as a *customization* that the organization may or may not be willing to implement for that customer, depending on business criteria. According to Krueger's model, the difference between a configuration and a customization is that the customization requires new code.

Salion adopted Krueger's reactive model. It first produced a "standard" product as its entry into the market. That product formed the basis for Salion's software product line and the basis for each new customer-specific product it fielded. The standard product was more than an engineering model from which "real" systems were produced; it gave Salion a very enviable entrée capability into customer organizations (see Section 2.7).

The rest of this section describes in greater detail Salion's software product line approach in terms of the product line practice areas [Clements & Northrop 02] that are especially germane to it. Each practice area is categorized as software engineering, technical management, or organizational management. Rather than organizing the practice areas by category, we organize them as follows to better show the evolution of Salion's product line:

- Scoping
- Understanding Relevant Domains
- Process Definition
- Architecture Definition
- COTS Utilization
- Tool Support
- Marketing
- Customer Interface Management
- Operations
- Data Collection, Metrics, and Tracking
- Structuring the Organization

We begin by describing one of the most key aspects of the Salion story: how Salion's reactive model gave it a way to handle unforeseen variations through partially *reactive* product line scoping.

3.1 Scoping

A product line's scope is a statement about which systems are in and out of the line's declared area of business. For organizations that take a proactive approach, their initial scope consists of the description of a set (more often, a *space*) of systems that will constitute their product line.

Salion was unwilling to *a priori* rule any system out of its product line's scope; a small company finding its legs cannot afford to turn down business. Moreover, in any "green field" approach, a firm scope definition would be very risky. Salion instead began with an informal scope definition based on the variabilities outlined in Section 2.4—a list that itself has grown over time. As customer products were added to Salion's repertoire, a *de facto* scope definition has emerged. As more products have been delivered, Salion's confidence in its original product space has grown. Now the scope revolves around the following variations that will cause the company to engage in customization work:

- forms and screens
- reports and exporting data
- bulk load requirements

These variations represent the three ways in which Salion expects to customize products. On the other hand, workflows, for instance, also remain a point of variation, but do not require customization. Rather, Salion's software is *configurable* with respect to workflow. Salion does not regard workflow as a dimension of its product line scope, because practically no work is required to achieve that variation.

And now, Salion's reactive scope definition has come to play the same primary role as a proactive scope would—it defines the company's business area. For example, it provides the marketing department with cues about how to engage the customer (see Section 3.7). The marketers now ask customers what their requirements are in the three areas listed above, whereas before, the marketers were much more likely to engage in an open-ended discussion about requirements. The scope also gives strong hints about what's "out." For example, when an early customer wanted to have a new kind of domain object modeled, Salion was quick to accommodate the request. Now, however, Salion is more likely to push back and try to frame such a requirement in terms of the standard (OAG) business object model that it has adopted.

3.2 Understanding Relevant Domains

In Salion's case, there was no long history of application domain expertise from which to precisely define anticipated products. In many ways, its products chart new territories. However, despite the newness, it did not proceed without domain investigation. Its founding organization, McKinsey & Company, turned over to Salion its extensive research into the manual management processes of custom and semi-custom manufacturing organizations that rely on bids and proposals to win business. This research provided not only the necessary market analysis for Salion's business proposition but also a domain knowledge base in acquisition management. Salion also drew on its own experience in related domains such as workflow and process enactment. In addition, several key people at McKinsey & Company had prior experience in ASP companies—a model that would come in handy at Salion.

Given this foundation, there is considerable evidence of the high degree of domain understanding that Salion accrued and modeled. In particular, it

- developed a rich set of business use cases, which has become Salion's *de facto* domain model and has remained relatively stable
- defined core schema that has remained exceedingly stable
- developed a Customer Business Object Model—a spreadsheet of objects and attributes that includes such information as where the objects are located in the database, what they are called in the user interface, and in which screens they are used. This model enables the architect and database administrator to build what is necessary. It has become so mature that it is now an artifact they send out into the field to be "filled in."
- embraced the OAG model, which has proven to be a very sound choice for its application market

The "Understanding Relevant Domains" practice area is likely to be a source of risk for new organizations, especially if they are venturing into new application areas. Salion seems to have successfully mitigated that risk.

3.3 Process Definition

A strong software process played a role from the beginning. Salion knew that the complexity of an unknown number of customers requiring an unknown number of variations would threaten to spin out of control of the company's small staff. So it put some strong process safeguards in place.

First, Salion decided to adopt the Rational Unified Process® (RUP®) to guide its development activities [Kruchten 00]. It overlaid that with the VRAPS (vision, rhythm, anticipation, partnering, and simplification) approach [Dikel et al. 01]. VRAPS represents an approach to software development that transcends tactical issues (such as what objects to create) and instead emphasizes strategic concerns such as keeping everyone in the organization aligned, dealing with change events, making sure that the right people are involved, keeping the ar-

® Rational Unified Process and RUP are registered trademarks of Rational Software Corporation.

chitecture simple, and establishing a regular pulse-like release schedule in concert with market conditions.

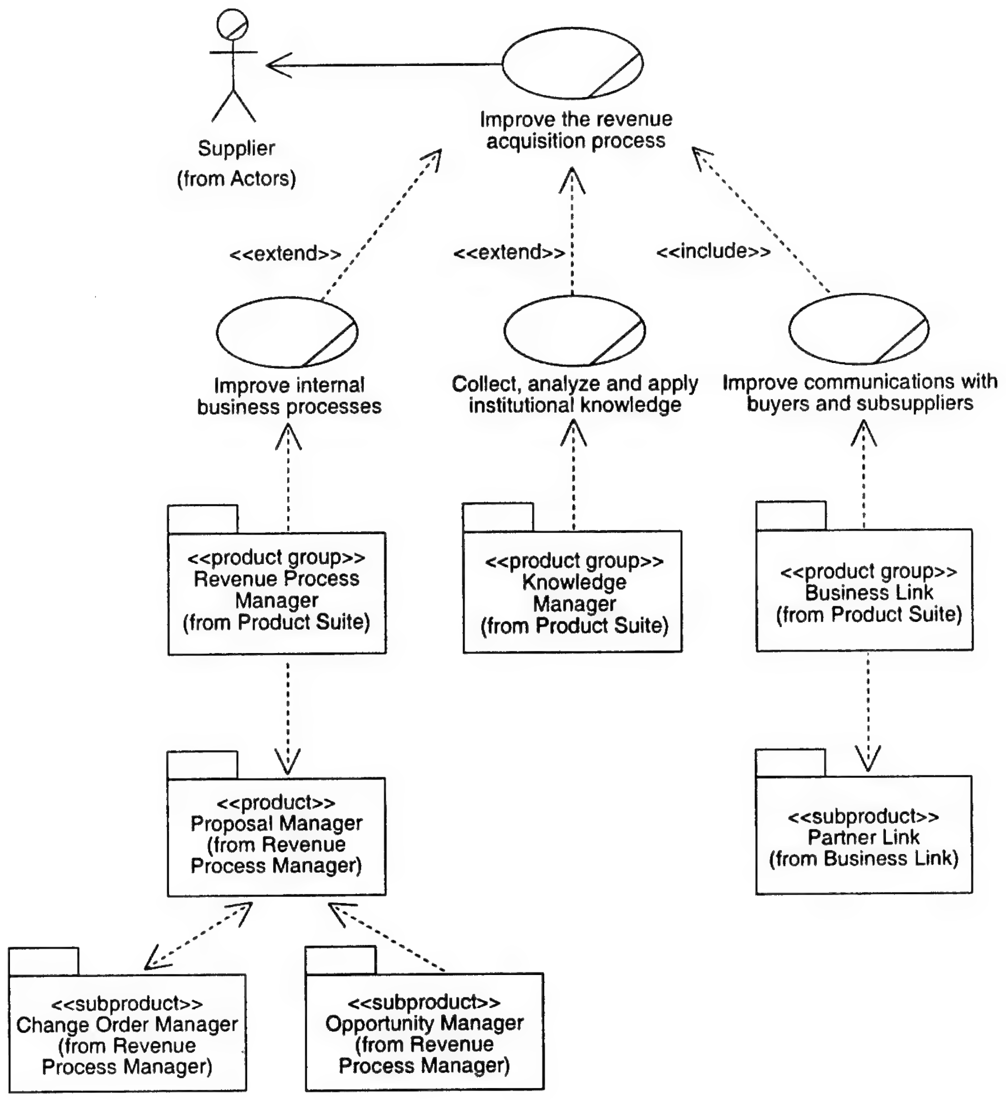
Into that mix Salion added some aspects of Extreme Programming [Beck 00]—namely, it carries out twice-daily builds, performs constant unit testing, and uses contracts as the interface among units. Also, every month it refactors its system's design to identify new commonality.

Keeping everyone aligned with the company's vision is a central theme of Salion's story. Salion applies RUP to write use cases not just for the company's software, but also for its business.

Figure 2 illustrates the Salion software solutions that realize the business use cases defined in the business use case model. Each product is represented as a package of functionality that provides extensions or modules that can be turned on or off, depending on the subscription level of each customer. This way of thinking permeates the business.

The business use case model sets the strategic direction of the company. Salion uses it to gauge the attractiveness of prospective clients. It keeps everyone on the same page in terms of knowing in which businesses the company is—and is not—engaged.

This capacity for keeping a group of people working synchronously toward a common goal is the essence of strong process definition and discipline.



Supplier
(from Actors)
Improve the revenue acquisition process
<<extend>>
<<extend>>
<<include>>
Improve internal business processes
Collect, analyze and apply institutional knowledge
Improve communications with buyers and subsuppliers
<<product group>> Revenue Process Manager (from Product Suite)
<<product group>> Knowledge Manager (from Product Suite)
<<product group>> Business Link (from Product Suite)
<<product>> Proposal Manager (from Revenue Process Manager)
<<subproduct>> Partner Link (from Business Link)
<<subproduct>> Change Order Manager (from Revenue Process Manager)
<<subproduct>> Opportunity Manager (from Revenue Process Manager)

Figure 2: Salion's Business Use Case Model

3.4 Architecture Definition

Every successful software product line depends on the right architecture. For Salion, this means an architecture that blends general concerns about runtime efficiency and protection of confidential data with production concerns about taking advantage of product line commonalities.

Salion's system architecture, shown in Figure 3, provides three tiers: (1) a front-end tier on top, (2) a core platform in the middle, and (3) a persistence layer (database) on the bottom.

The front-end tier consists of a pair of JavaServer pages™ (JSP™) servers, to provide redundancy for increased reliability. Hosted systems (e.g., Salion hardware running at Salion's site) share the core platform and persistence layer, but each customer gets a dedicated pair of front-end servers with a unique uniform resource locator (URL) for access. This is done to provide high availability and prevent data confidentiality breaches across customer systems.

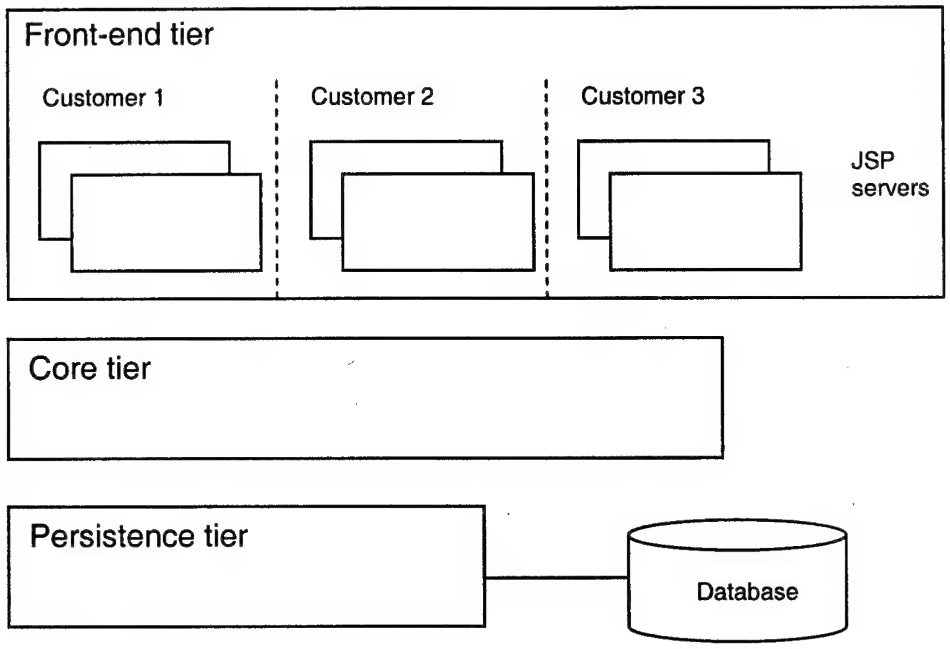


Figure 3: Salion's Product Line System Architecture

Customized software runs only on the front-end servers; the other platforms run common software. The front-end servers are currently simple and inexpensive Solaris boxes. The other hardware consists of about 10 "really beefy" (in the words of the architect) application servers that are responsible for carrying out the systems' business logic and core (common) applications, as well as hosting Web servers, the network infrastructure, and the security authentication subsystem.

This system architecture, and the way that the software is allocated to it, makes upgrades and customizations straightforward. If customers want a small upgrade to their customized software, Salion can provide it easily. If customers want an upgrade that will tax the capabilities of the front end, Salion can scale up that area without affecting any other part of the architecture or any other customer. A major upgrade that has widespread functionality or performance requirements can be done in concert with an upgrade to the application servers.

™ Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

The software architecture is decomposed to be compatible with the deployment scheme just described. Also, the availability and use of commercial off-the-shelf (COTS) components dictate additional decomposition criteria (see Section 3.5). Variation points are implemented using the Façade design pattern [Gamma et al. 95].

The architecture for the product line is a firm by-product of the VRAPS approach's simplification principle. The chief architect has adopted a set of strong, sensible principles to ensure that the architecture is no more complex than is necessary.

As mentioned previously, security and availability lead the list of required attributes. Seamless 24/7 operation is a requirement, as is the unconditional preservation of customers' data confidentiality. These mandates precipitated the following foundational architectural decisions:

1. The architecture is stateless. The system can always recover no matter what it is doing when an upset occurs.
2. There is no single point of failure in the architecture. Salion uses a redundant network and a replicated Oracle database running in hot standby mode. The system is by and large composed of a set of identical homogeneous servers, and each server is either in service or out of service.
3. Salion uses COTS components when possible, but does not hesitate to build its own components when necessary or to mitigate risk (see Section 3.5). The Java 2 Platform, Enterprise Edition (J2EE™) and the Common Object Request Broker Architecture (CORBA®) provide the underlying platform, but Salion's architect imposes guidelines for using Enterprise JavaBeans™ (EJB™) to enhance reliability. One job of the company's database administrator is to control the queries sent to the database—writing tight queries and monitoring the Structured Query Language (SQL) that developers write.

Optimizations such as the use of careful SQL provide the key to performance and are chosen carefully. The architect takes responsibility for targeting the optimizations that will bring about real improvement and avoiding those that do not address a demonstrated need. For example, early on, the architect gathered data about where cycles were being spent in the architecture, namely, in Java Naming and Directory Interface™ (JNDI) lookups, object invocations, and static hypertext markup language (HTML) traffic. Armed with this data, the architect implemented four minor optimizations that quadrupled the system's throughput and provided a performance cushion that still holds. This is in contrast to an "optimize everything" approach that consumes developer resources and may not provide a return in runtime benefits.

® CORBA is a registered trademark of Object Management Group, Inc.

The result is a software architecture that consists of four logical layers:

1. presentation layer that provides a unified interface for accessing all the product's applications. This layer is devoid of business logic and corresponds to the "view" part of the Model-View-Controller pattern [Krasner & Pope 88]. Clients access this layer through a Web browser from an intranet or the Internet via hypertext transfer protocol (HTTP) requests. The software in this layer is HTML and JavaBean driven. It channels user requests to the appropriate calls or servelet invocations in the lower layers.
2. business applications layer that comprises EJB components with session facades from the J2EE framework. Business logic resides in this layer.
3. core services layer that provides the reusable subsystems, such as those for document and workflow management. This layer contains software for connecting enterprise resource planning (ERP) and continuous risk management (CRM) systems, and for functionality for importing existing data on customers' systems. The OAG business object model also resides here.
4. persistence layer, where the Oracle database and Borland's Java database connectivity (JDBC^{TM}) database drivers reside

In addition to those four layers, a unifying security framework exists alongside them. It protects the applications and data at each level by providing authentication and security services.

A typical product consists of about 40 independently built submodules totaling some 150 thousand lines of code (KLOC).

The goal of a product line architect is to provide a development and runtime environment in which developers don't have to worry about debugging cache, SQL queries, or transaction protocols—in short, to let the developers concentrate on the application's business logic. The architect has tried to instill an architectural culture within Salion, based on the following principles that ground the nimble, flexible architecture:

- Understand and capture the rationale behind every design decision. The goals are to know why you want to do something *before* you do it and to be able to look it up afterward. For example, why did we use a COTS component here, but not there?
- Solutions are not needed when heuristics will do. This philosophy has manifested itself in a reluctance to implement a complex generic solution to something when it is easier to write a small bit of custom code to handle each situation as it occurs.
- Instead of letting important information and experience slip away, Salion's culture encourages their capture in structured repositories that can be accessed later.
- Provide scalability and eliminate complexity. For example, there is no cache in the architecture because it is unnecessary. CVS is the configuration management tool of choice because it's free, everyone knows how to use it, and it's far simpler than the gold-plated industrial-strength configuration management (CM) tools that many organizations use.

This attempt to instill an architectural philosophy throughout the company is another example of how Salion wants to make sure that the entire staff shares a unified vision. If architectural stakeholders hold the same vision as the architect, they will come to understand which kinds of changes are feasible, which are likely to be met with resistance, and (most importantly) why the decisions to make them support the organization's business model.

3.5 COTS Utilization

Salion's philosophy about using COTS components reflects its deliberate commitment to simplicity, but not at the expense of quality. Salion's chief architect and vice president of engineering are both former employees of a major Web search engine site/content provider. In four years, they watched the site/provider's usage go from 45,000 to 45,000,000 page views per day. With millions of people using the system, they learned very quickly to do what it takes to avoid being awakened in the middle of the night with a business-threatening problem.

Salion's major concern about COTS products centers on its ability to provide the necessary quality attributes (i.e., performance, security, and reliability) and the product's ability to scale. When the product's ability is in doubt and no practical workarounds are available, Salion will not hesitate to build the necessary components in-house.

Salion's rule of thumb is that if the function is unimportant, COTS will do. If there's an actual or *de facto* standard for some aspect of the system, COTS will do (as there is likely to be a choice of more than one vendor that meets the standard). Otherwise, that component is a strong candidate for in-house implementation.[4]

An example is the commercial workflow system that Salion adopted initially. After discovering that the system didn't have the necessary flexibility, scalability, or reliability, and that it didn't obey any applicable standard, Salion jettisoned it in favor of a homegrown version.

3.6 Tool Support

Tool support is an area where Salion has shown a willingness to be innovative. In addition to the usual array of software development tools, Salion employs two emerging product-line-specific tools: Gears (<http://www.biglever.com>) and CloneDR (<http://www.semanticdesigns.com>).

[4] This defined decision process also qualifies as a practice in the "Make/Buy/Mine/Commission" practice area.

Gears is designed to permit organizations to manage the commonality and variability present in its product line assets. Gears works by identifying artifacts (e.g., source code files) that are used in every member of the product line and artifacts that are used in some, but not all, members. Then it facilitates the build process for a particular product by retrieving the assets that belong to that member and making them available to the normal build tool.

The advantages of Gears are four-fold. First, it obviates the need for complicated layers of `#ifdef` statements in the code that quickly become impossible to read and understand in all but the simplest cases. Second, it allows a vastly simplified build process that can now be identical for every member of the product line. Third, because it is file based, it can work for non-code artifacts such as design documentation, user documentation, test cases, data files, work schedules, and so forth. And fourth, it reinforces a fundamental concept in product line development—that the entity being built is not a set of products but a single product line.

The last point is subtle but fundamental. Product line organizations with the most success view their endeavor not as building an ad hoc set of products that might have things in common, but rather as developing and nourishing a strong singular product line. Gears reinforces this by providing a single venue for building every product, by explicitly identifying common artifacts and product variants, and by mapping artifact variations to products.

Operationally, Gears lets Salion run two variants at the same time without having to worry about "breaking" Salion's standard product.

Currently, Gears helps Salion manage a total of 3,333 files, 88 (2.4%) of which represent variations. Put another way, 97.6% of the files are reused (i.e., do not change) across products.

CloneDR is a tool that provides clone detection—places where the code is almost similar and can be refactored to extract the commonality. CloneDR uses sophisticated algorithms that go far beyond simple syntactic comparison. Salion found that some 13% of its stable code was essentially duplicate and could be removed and combined after some simple reengineering. In one case, the removal of three clones eliminated some 27 KLOC.

Together, Gears and CloneDR provide a naturally synergistic capability for generating a product line from two or more already-fielded systems that were developed from each other. Identical clones become common components; almost-alike clones and product-specific software define dimensions of variation to Gears, which can then quickly cast the products as variations of a single product line.

3.7 Marketing/Customer Interface Management

In the beginning, Salion employed a "solution selling" strategy to connect with its customers. This strategy involved performing a needs assessment for a customer and then selling to those needs. The needs turn into business use case models, following RUP, at a two-hour meeting held every Wednesday. This so-called "joyous vision" meeting is an executive-level discussion among the chief financial officer (CFO), the vice president of marketing, the vice president of engineering, the chief architect, and others at this level to choose and prioritize among business opportunities, all under the auspices of Salion's own business use case model that drives the company. Salion still employs this approach, but as mentioned in Section 3.1, is much more inclined these days to channel the customer's statement of needs into the three main dimensions of variation inherent in its product line.

Thus, like most product line organizations, Salion's marketers do not have carte blanche to sell whatever the customer wants. Rather, they must first receive approval or confirmation that a proposed system is in the product line's scope. Of course, they try to sell each customer a basic system first (one that can satisfy the customer's needs through preplanned configuration) or, failing that, a variant already fielded.

The first litmus test applied to a new customer need is whether it's in Salion's business use case model. For instance, a customer who wanted to buy Salion's workflow engine would likely be turned down, because there is no use case that says Salion is a component provider.

If the customer's needs can be met only by a new system, Salion first predicts the cost based on past performance (see Section 3.9). At that point, whether to achieve the requirements by customization (new code) or configuration (expanding variabilities already in place) is a technical question that must be answered. But its answer is linked to business questions dealing with costs, the likelihood of future use, and which strategy is likely to make Salion more money. Thus, Salion's marketers must walk a fine line between keeping customers happy by selling features and limiting promised customizations.

But as a start-up company, Salion has been loathe to turn down business, and being able to say "yes" to a customer right out of the gate is an enormous advantage. Because of its flexible architecture and robust development process (see Section 3.8), Salion can say "yes" most of the time.

Also contributing to this ability to say "yes" is an innovative marketing arrangement made possible solely by the product line approach. All implementation work for a customer occurs in two phases. During the first phase, Salion installs its standard system for a customer within 60 days. During the second phase, Salion builds and installs a customized product. While the standard system is running, Salion works with the customer to define customizations and then, based on this elaboration step, commits to a delivery date and (fixed) contract price. This two-phase approach has many benefits:

- It makes customers happy. In a fixed, relatively short period of time, they can begin receiving benefits from the system.
- Having a working system in place allows Salion to work with the customer to define, elaborate, and specify customizations in a low-pressure setting.
- It gets the Salion system in the door, providing the customer with most of the eventual functionality in a short time. Salion regards this capability as a big part of saying "yes." The remainder of saying "yes" can be deferred until the second phase.
- Having a working system in place lets the customer define desired customizations more intelligently, in the context of what is available and feasible. The customer speaks the language of the system, so to speak, as opposed to asking for arbitrary features with no operational pedigree. The customer becomes more of a partner.

The customer's primary interface with Salion is through its Professional Services group, that configures delivered systems, initiates customizations, and (in the case of one-time-use-only components) might actually build custom parts. Salion understands that its customers just want a system and don't want to be bothered with configuring it. So that's what Salion delivers, and whether customers get their systems through customization or configuration is a detail Salion can keep hidden and over which it can maintain flexibility.

As mentioned in Section 3.1, the marketing group gently steers the customer to three areas of customization: (1) forms and screens, (2) export and reporting, and (3) bulk load (i.e., legacy data migration) requirements. Other variations are possible, but they are done via the configuration of built-in preplanned variation mechanisms, and so do not represent an area of intense concern to the development group.

3.8 Operations

The "Operations" product line practice area is concerned with the day-to-day running of the product line and how various stakeholders come together to cooperatively make sure that new products in the product line are launched, built, and deployed smoothly.

3.8.1 Normal Operation

Salion's operational strategy manages three different product cycles at any one time. While one product is in the inception stage, another is in requirements design and elaboration, while a third is in development.

The business use case model is the driving impetus for all three stages. This leads to other use case models for a given product, subsystem, or sub-subsystem. Those models are standard

RUP practices, but as mentioned previously, Salion has also adopted some of the tenets of Extreme Programming. These tenets include Salion's twice-daily builds, emphasis on unit testing, use of contracts as the interface among units, and monthly refactoring to identify new commonality (which the Gears tool helps manage).

Salion's "rhythm" part of the VRAPS process comes from a planned release every 30 days. At that interval, a packet is given to the user-interface developers working on the front end and to the developers working on back-end components. There is one packet per component, and typically 5 (out of the 40) are being worked on at a time. A packet contains any of the following, as needed:

- list of applicable use cases
- high-level use case models
- data models (for the database administrator)
- class diagrams
- user-interface page flow
- user-interface mock-ups
- database schema

A new release might contain enhanced generic properties or features, or represent a new variation contained in a new member of the product family.

After coding is complete, the system goes into quality assurance (QA) and testing. Most of the process after this point is scripted, handling checkout from the CM system, the build step, and the application of a suite of some 350 white-box tests. All told, this process takes about two hours.

To keep problems from derailing the clockwork rhythmic process of development and release, a so-called "joyous love" meeting is held every Tuesday and Thursday to address implementation-level concerns and keep projects on track. The objectives of these meetings include

- Come up with a contract between owners of front-end components and owners of back-end components. The contract could be a written interface or an informal agreement.
- Resolve any communication or design conflicts.
- Make sure that everyone is on track. In the words of one participant, they "will stop heaven and earth to resolve issues." The idea is to do what it takes to make sure that no one is being held up.

At end of each release iteration, the chief architect and some of the key developers have a "quiet week." They run scripts that show coupling/cohesion among modules and inspect code. They post a list of recommendations, which are addressed in the following release, after being prioritized in the next "joyous love" meeting.

The QA team, by contrast, is never idle and always working to maintain and improve the quality of the artifacts.

3.8.2 Introducing New Variants

If a new product is given the go-ahead, the architect must decide how to handle the new requirements. Depending on several factors, the requirements might

- become part of the new core capabilities
- be built as core capabilities, but be added as part of a new licensing option
- be built as a customization, but be included in the product builder's guide as a common customization request

At that point, developers are assigned to make the necessary changes to the various artifacts. In a pilot demonstration of the product line concept, developers were tasked with building a new variant that produced a special kind of report detailing the history of the kinds of bids the supplier lost. This change resulted in adding new business logic to that layer of the architecture and required about two days of effort. Out of almost 3,000 classes, only about 20-30 required modification.

3.8.3 Phasing Out Old Products

To simplify its life-cycle support costs, Salion prefers not to maintain old versions of products. Where possible, this provision is built into Salion's contracts. When a customer product undergoes maintenance, the first inclination is to upgrade that product to the current version of all software, if possible.

3.9 Data Collection, Metrics, and Tracking

Salion uses metrics as a key planning tool. At the time of our first visit, it had accumulated 12 months worth of data and heuristics that help it predict with confidence what it can do by when.

Use cases are used to drive productivity measures. Numbers that are lower than expected help Salion spot potential problems, such as wheel spinning due to inadequately specified contracts.

For each release, Salion measures the

- planned number of use cases implemented
- actual number of use cases implemented
- release date
- new use cases found
- use cases per staff
- number of bugs found
- number of bugs fixed
- number of JSP pages created
- number of JSP pages modified
- ratio of engineers to marketers used

These measures help Salion determine its capacity for new work and prevent it from promising more to a customer than it can likely deliver. They also help to avoid "yes you can, no we can't" battles by simply establishing an historical baseline and allow Salion to answer questions such as how much work it could do if it doubled its staff.

3.10 Structuring the Organization

Every software product line organization must adopt an organizational structure that promotes the most advantageous production process. Although other variations are possible [Bosch 00], fundamentally there are two organizational structures for product lines that differ in who produces the core assets that are reused across all the products in the family. One model establishes a separate core asset group (sometimes called the "platform" group) that is separate from the groups that turn out products. The second model assigns responsibility for building and maintaining the core assets directly to the product groups.

Salion is small enough—with only 10 people involved in product development—that crafting a formal organizational structure is not paramount. However, it is still instructive to examine how Salion addressed the question that every product line organization must face: who builds the core assets?

Early on, before Salion sold its first product, it worked on building its so-called "standard system." This was the system that would form the basis of Salion's reactive software product line model. Future systems would be built from this one, and it is still the system that is installed initially for a new customer. Since a reactive model proceeds largely without knowing in advance what is going to be reused, all the assets were regarded as at least potentially core. During this start-up period, Salion's three-cycle operational approach (see Section 3.8) drove its organizational role assignments. There was no separate core asset team; everyone worked on producing successively more capable versions of the product.

This model has persisted even with Salion's third customer product under way. When Salion needs to make a major customization, a product-based team is assigned to make it happen.

4 Payoffs and Benefits

Our interviews at Salion were conducted in two phases, separated by nine months. During the first phase, Salion had not yet landed its first customer, and many of its plans and approaches could be characterized as tentative and hopeful. During our second visit, Salion was working on a system for its third customer, and the difference in tone was striking. Confidence in its approach was high, and speculation was replaced by assertion. Salion is now in a position to recognize and recount the significant benefits it has reaped from its product line approach, some of which we describe here.

First and foremost, Salion can produce and support sophisticated, highly secure, high-availability, COTS-intensive systems with a minimum of staff. Currently, there are 21 people in the company. Seven are counted as developers, plus one who is in charge of QA and one who is in charge of the build process. As the first round of interviews for this report was being taken, Salion had produced its 12th in a series of 30-day releases, all of which were on schedule.

Salion's product line approach lets this small staff work very productively. Building the standard product took 190 person-months. Building the first customer product took just 15 person-months and achieved a phenomenal 97% reuse level. Projections for the second customer product are for 30% less effort still. This second product is on track and expected to be delivered "with room to spare."

From a business perspective, Salion's product line approach helps it deal with its investors. Every start-up has to deal and struggle with growth. Venture capital can get a company up and running and serving its first few customers, but of course the end game is for the company to become self-sustaining with a very large customer base. So, the people funding the start-up ask "How are you going to scale?" Usually the answer involves rewriting the product to make it more robust, increasing the development and QA staff by some integer multiple, and moving into a larger office space. Salion has a much different answer: it is poised to grow (in terms of its customer base) exactly as it is. Its organization, process, and architecture—in short, its product line approach—provides it with vast untapped capacity. This capacity makes Salion a very attractive place for its investors to put their money, because the company features a low "burn rate" compared to conventional organizations. As Salion's chief technical officer (CTO) put it, in the next round of funding, the company will be able to ask for half as much and do twice as much with it. In its third major round, it expects to be supporting 10-15 customer products with a staff that is still tiny by comparison to non-product-line-based software organizations.

5 Conclusions and Lessons Learned

This section synthesizes some of the major lessons we learned from Salion's experience with the software product line approach.

5.1 Joyous Software Development Process, Part 1: Unified Vision

One of the overriding themes that makes the Salion story compelling is the company's devotion to making an overall business plan, letting the plan percolate down through the organization, and having everyone adhere to it in spirit and in practice. In short, everyone at Salion is on the same page. The executives have succeeded in instilling everyone with the business objectives and producing a unified organization in which everyone is working toward the same goals. The architects understand beyond a doubt that the architecture is responsible for providing the capabilities required to carry out the business plan. In turn, they have succeeded in instilling architectural principles in the group that supports that plan.

This unified view is certainly made easier because of Salion's small size, but it would be a mistake to imagine that this was all there was to it. Salion's managers spend a great deal of effort putting people on the same page, and they do it at three levels:

1. "Joyous vision" meetings, detailed in Section 3.7, are held every Wednesday among executive-level decision makers to choose and prioritize business opportunities.
2. "Joyous chunk" meetings, held regularly in the context of a product contract, enable the organization to plan the customization and configuration that will be required to meet a particular customer's product needs.
3. "Joyous love" meetings, described in Section 3.8, are held every Tuesday and Thursday to handle implementation-level issues that might otherwise threaten to derail or delay product-building efforts.

In addition, the proactive inspections that the chief architect takes upon him/herself help assure high architecture and process conformance.

An example of how the business and technical sides of the house work together seamlessly is the company's option-licensing strategy. Recall from Section 2.3 that each system Salion ships is a suite of individual products from which its customers can pick and choose. Early on, Salion had to decide whether to ship a system to a customer as a set of the chosen products or as an entire system with only the chosen products enabled. This decision had business and technical ramifications. The marketers loved the entire-system approach, because it made it easier for them to sell license "toggles" to turn on new products. It was also appealing technically, because it was simpler: each customer was shipped the same system. Building, installation, maintenance, and upgrades were vastly simplified, and that appealed to the CFO.

The decision to ship the total system was not by itself particularly controversial, but the process for making it shows how business, financial, marketing, and technical people at Salion all speak exactly the same language and sit at the same table.

Salion's devotion to its business plan imbues developers with a sense of ownership over the company's components and makes developers conscientious about stabilizing those components for reuse. Each component is important for the success of the company's product line.

5.2 Joyous Software Development, Part 2: The Best of All Worlds

Though certain essential activities and practice areas are common to any successful software product line effort, we don't advocate any specific, prescribed methodology because many practices are relevant for each practice area. Each organization needs to choose those practices that fit its unique context—a context that encompasses culture, talent, business goals, and the product line approach.

Salion has been both selective and pragmatic about assembling the specific practices to effect its product line business and technical strategy. It has chosen with agility and care best-of-breed techniques and then pruned and combined them to its advantage. Salion's approach to software product lines is a unique and successful intertwining of techniques and principles from RUP, VRAPS, Extreme Programming, and the SEI with the added automated support provided by Gears.

Salion has evolved its most effective blend of practices for software product lines with a true continuous improvement mentality. Throughout, it has religiously maintained a strong adherence to process discipline. When Salion hacked, it regretted it, and that regret strengthened Salion's resolve to be more disciplined about following its defined processes. Recently it conducted a Capability Maturity Model® (CMM®) self assessment with a resultant "almost Level 3." Salion has resolved to make the changes necessary for it to be a veritable Level 3.

® Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

5.3 Customization Versus Configuration

The conventional wisdom in software product line production is that the core assets should be tailorable using built-in variation mechanisms that render the components generic over a wide range of application instances. Component generators exemplify this philosophy; the use of configuration parameters is a widely adopted variant. The appeal is obvious. Product building becomes a matter of "generate and integrate," with a minimum of source code writing.

Salion knew from the beginning that its products would have to be variable over several dimensions. For example, every customer was going to want to use customized forms, so Salion set about writing generic forms software. Given this infrastructure and a statement of a customer's reporting needs, Salion could quickly generate or instantiate a forms component that would satisfy the requirements.

What sounded like a good idea turned out in practice to be a keen disappointment. After eight months of development time, Salion had a generic forms module that didn't work, was much too overblown for what was actually needed, and failed to help the company meet the real requirements of its customers. Recently, the whole thing was thrown out.

Salion experienced the same disappointment and frustration with customized reports. Trying to build a generic component in the absence of actual customer requirements resulted in wasted time and a technical failure.

These and similar lessons have given Salion a completely new outlook on achieving variation. Now its first inclination is to provide variation through customization, not configuration. This means that Salion will modify or rewrite the necessary core assets to produce a version of the system that satisfies a particular customer. As it gains more knowledge about a particular domain, it may eventually be able to design a configurable (or even generative) infrastructure for the domain. For instance, of the three major variation points of Salion's products (forms and screens, reporting and export, and bulk load), the company suspects that there's a framework lurking in the first two, which may one day be teased out in a practical fashion. But for now, customization is the watchword.

Many other companies also produce new products by simply changing what they have, but they quickly run into a wall of complexity while trying to manage the combinatoric variations present in separately evolving systems. This occurs because they clone the entire system to produce new versions. Salion never loses sight of the fact that it is producing a single product line with a small number of variations. Hence, it can keep tight control over the vast majority of the parts that are common to all family members. Undoubtedly, the Gears tool plays a large part in maintaining this control, both intellectually and technically, as it was designed explicitly to serve this purpose.

Another example of an ill-fated configurability effort came when Salion discovered that its first customer did not use the OAG standard object data model that Salion (and much of the automotive industry) had adopted. After a six-week analysis, Salion decided that the right solution was to build a configurable data item subsystem, so its products could quickly adapt to whatever object models a customer was using. Now, in retrospect, Salion is trying to eliminate this subsystem, because subsequent analysis has revealed that in fact the OAG model would have done nicely after all—it turned out to be possible to map all 145 features in this customer's data model right to the OAG framework.

5.4 Risk Management Culture

A new organization with no marketplace experience in its chosen application domain runs a risk in embracing a product line approach. Salion further exacerbated this risk by selecting previously untested combinations of approaches—namely, RUP, Extreme Programming, VRAPS, and Gears. However, at no point has Salion been blind to the risk it is assuming. Instead, it has embraced a rigorous set of risk management practices that are informed by the metrics it collects.

Salion mitigated the risk of its "green field" product line by proactively arriving at a requisite level of domain understanding and by choosing a reactive approach to product lines. With its series of "joyous" meetings at multiple levels, Salion engages in risk analysis and introspection that has led to risk mitigation strategies for undercutting what could develop into costly problems. Its total approach is a well-balanced act of innovation coupled with risk management.

Moreover, Salion has capitalized on its very nature as a small start-up unshackled by the cultural barriers and ingrained infrastructure that are so hard to dismantle in larger, more established companies. Salion recognized that it can shape its culture as it goes and has been nimbly doing so, all the while skirting risks that could prove to be land mines.

6 Conclusion

The fact that Salion has the following things in common with other successful product line organizations underscores the importance of those things being common denominators of success:

- Salion has a need for a successful software product line approach that is directly connected to its success as an emerging company.
- Salion has several product line champions. It is an organization that strives mightily (and successfully) to make all its organizational departments speak the same language, share the same vision, and strive for the same goals.
- Salion has a strong architecture-driven approach, a talented architecture team, and a robust, flexible product line architecture. Salion's chief architect provides strong and clearly articulated principles (such as avoiding complexity wherever possible) that are derived from the need to satisfy the company's business plan.
- Salion has an involved and committed management team that proactively supports the product line approach.
- Salion is an organization that is not only comfortable with but thrives on a high degree of process discipline.

In addition, there are other characteristics of this organization that point to a successful software product line:

- Salion's team drives for best practices and excellence in software development.
- Salion enthusiastically embraces metrics and tracking to help it manage its production, uncover areas of high-payoff improvement, and self-diagnose.
- Salion fosters a culture that is not afraid to try new things, such as state-of-the-art tools or an innovative best-of-breed mix of three different process paradigms.
- Salion is a self-aware organization that welcomes critical introspection and thoughtful review.

Salion's software product line story is the story of a small, nimble organization that from the beginning recognized that a reactive product line approach was the way to achieve flexibility in an application domain in which the future could not be predicted reliably. How Salion achieved success with this model, evolved its scope, became more concrete over time, skillfully managed an innovative process model, and gathered fundamental insights about customization versus configuration are all parts of Salion's unique story.

7 For Further Reading

Other SEI software product line case studies have detailed the experiences of

- CelsiusTech, a Swedish defense contractor that adopted the product line approach for its family of very large real-time embedded naval ship systems [Brownsword & Clements 96, Bass et al. 98]
- Control Channel Toolkit (CCT), a satellite ground control systems' asset base procured by the U.S. government [Clements & Northrop 02]
- Market Maker Software AG, a small German company that builds financial Web sites for major banking institutions [Clements & Northrop 02]
- Cummins, Inc., the world's largest manufacturer of large diesel engines, that used the product line approach to parlay 20 basic releases into over 1,000 separate products and achieve a dominant position in the industrial diesel engine market [Clements & Northrop 02]

Other SEI reports illustrate the product line approach from the point of view of a government acquisition agency [Bergey & Goethert 01] or concentrate on particular technical aspects of fielding a product line. Those aspects include building a business case [Cohen 01] and carrying out architecture reconstruction [O'Brien 01]. In addition to these publications, the proceedings of the first two international Software Product Line Conferences (SPLC1 and SPLC2) include many papers that contain detailed experience reports from organizations that have adopted the product line approach [Donohoe 00, Chastek 02].

Clements and Northrop offer a comprehensive treatment of the software product line approach, including a description of the 29 software engineering, technical management, and organizational management practice areas [Clements & Northrop 02].

References

[Bass et al. 98]	Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice.* Reading, MA: Addison-Wesley, 1998.
[Beck 00]	Beck, K. *Extreme Programming Explained.* Reading, MA: Addison-Wesley, 2000.
[Bergey & Goethert 01]	Bergey, J. K. & Goethert, W. B. *Developing a Product Line Acquisition Strategy for a DoD Organization: A Case Study* (CMU/SEI-2001-TN-021, ADA395202). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. <http://www.sei.cmu.edu/publications/documents/01.reports/01tn021.html>.
[Bosch 00]	Bosch, J. "Organizing for Software Product Lines," 117-134. *Proceedings of the 3rd International Workshop on Software Architectures for Product Families (IWSAPF-3).* Las Palmas de Gran Canaria, Spain, March 15-17, 2000. Berlin, Germany: Springer-Verlag, 2000.
[Brownsword & Clements 96]	Brownsword, L. & Clements, P. *A Case Study in Successful Product Line Development* (CMU/SEI-96-TR-016, ADA315802). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996. <http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.016.html>.
[Chastek 02]	Chastek, G., ed. *Software Product Lines: Proceedings of the Second Software Product Line Conference (SPLC2).* New York, NY: Springer, 2002.
[Clements & Northrop 02]	Clements, P. & Northrop, L. *Software Product Lines: Practices and Patterns.* Boston, MA: Addison-Wesley, 2002.

[Cohen 01]	Cohen, S. *Case Study: Building and Communicating a Business Case for a DoD Product Line* (CMU/SEI-2001-TN-020, ADA395155). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. <http://www.sei.cmu.edu/publications/documents/01.reports/01tn020.html>.
[Dikel et al. 01]	Dikel D.; Kane, D.; & Wilson, J. *Software Architecture: Organizational Principles and Patterns.* Upper Saddle River, NJ: Prentice Hall, 2001.
[Donohoe 00]	Donohoe, P., ed. *Software Product Lines: Experience and Research Directions, Proceedings of the First Software Product Line Conference (SPLC1).* Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic, 2000.
[Gamma et al. 95]	Gamma, E.; Helm, R.; Johnson, R.; & Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley, 1995.
[Krasner & Pope 88]	Krasner, G. & Pope, S. "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80." *Journal of Object-Oriented Programming 1,* 3 (August/September 1988): 26-49.
[Kruchten 00]	Kruchten, P. *The Rational Unified Process: An Introduction,* Second Edition. Reading, MA: Addison-Wesley, 2000.
[Krueger 01]	Krueger, C. "Easing the Transition to Software Mass Customization," 282-293. *Proceedings of the 4th International Workshop on Software Product Family Engineering.* Bilbao, Spain, October 3-5, 2001. New York, NY: Springer-Verlag, 2001.
[O'Brien 01]	O'Brien, L. *Architecture Reconstruction to Support a Product Line Effort: Case Study* (CMU/SEI-2001-TN-015, ADA395167). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. <http://www.sei.cmu.edu/publications/documents/01.reports/01tn015.html>.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE November 2002	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Salion, Inc.: A Software Product Line Case Study		5. FUNDING NUMBERS F19628-00-C-0003
6. AUTHOR(S) Paul C. Clements & Linda M. Northrop		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2002-TR-038
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER ESC-TR-2002-038
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

This report is another in a series of Software Engineering Institute (SEI^{SM}) case studies of organizations that have adopted the software product line approach for developing systems. It details the story of Salion, Inc., an enterprise software company providing Revenue Acquisition Management solutions tailored to the unique needs of automotive suppliers. Salion's solutions enable suppliers to organize and manage their disparate customer-interfacing activities as one coordinated business process, resulting in higher revenues, profit margins, and customer satisfaction.

This case study is unique in that Salion did not have substantial experience in its application area, although its key designers and strategists were knowledgeable about related domains. Salion pursued a reactive approach to its product line that let it respond flexibly to spontaneous business opportunities and that significantly lowered the cost of adopting the product line paradigm to its software system development. This case study describes relevant dimensions of Salion's context, how it approached several product line practice areas that were key to its strategy, the benefits gained through its product line, lessons learned, and the major thematic aspects of the Salion story.

14. SUBJECT TERMS software product lines, product line practice, product line practice framework, product line case study			15. NUMBER OF PAGES 50
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL